Sandfly Safety and Performance

2024-07-29

Introduction

Sandfly is an agentless endpoint detection and incident response platform for Linux. Our product is known for high-performance, wide compatibility, and low risk of causing stability issues vs. traditional agent-based solutions. We are the no drama security monitoring tool for Linux

This document describes safety and performance considerations for Sandfly. While each network is different, these guidelines are a fair representation of what to expect when operating Sandfly and how we reduce customers' risk in our product. We also list several recommendations for making Sandfly scans more efficient and reducing impact on systems and networks even further.

Safety and Performance Criteria

Sandfly has strict guidelines about what we do and don't do that make our product significantly safer vs. traditional agent-based systems:

1) We never hook into the kernel as it can cause serious stability, performance, and compatibility risks.
2) We use multiple methods to limit CPU usage.
3) We never do full harddisk signature sweeps like traditional anti-virus, unless specifically directed by manual incident response operations.
4) We compress telemetry data to conserve bandwidth.
5) We have multiple failsafes to halt investigations taking too long on systems.

Safety and Performance Impact

System safety and performance are the #1 priorities for Sandfly. We take these issues very seriously and the product was designed on multiple levels to avoid causing system impacts. On Linux, the areas of potential impacts are the following:

- System Stability and CPU
- Memory
- Disk
- Network

We discuss below what Sandfly does to limit impacts in each.

System Stability and CPU

Sandfly does not tie into kernel space with modules, system call hooking, eBPF monitoring, memory sweeps, and similar tactics. As a result, we have almost no chance of causing system crashes or sustained high CPU loads like agent-based products.

Specifically, Sandfly incorporates the following protections to manage system stability risks:

- Avoiding the boot process
- Never hooking the kernel
- Single thread scanning
- Low process priority
- Data caching

Avoiding the Boot Process

Sandfly does not need to start software on boot such as persistent services, system drivers, or inserting kernel modules. We show up after the system is booted to do our investigations and then leave. There is no critical software path on remote systems that Sandfly inserts itself into that can cause mass boot failures or widespread kernel panics.

Additionally, Sandfly does not install software packages on remote systems. If by some small chance Sandfly were to be causing system issues, operators can simply disable the scans and any potential impacts are halted. There is never a need to undo software installs on each endpoint.

Never Hooking the Kernel

Agent-based solutions will hook kernel system calls or use eBPF to monitor the same in kernel space. While this is effective for a small number of use cases, it has extremely serious risks in terms of safety, stability, and performance. All methods of kernel space monitoring have a risk of causing a kernel panic and this is well documented in bug reports on the Internet.

Further, it is not possible to do kernel level monitoring without having performance impacts. In fact, the busier a system gets, the more performance risk kernel space monitoring has as it must do even more tracking of system activity.

Sandfly avoids stability and performance risks as we do all our operations in user space, not kernel space, which is **significantly safer**. It is a myth that you need kernel space monitoring to find attacks on Linux and it carries serious risks for stability, performance, and compatibility doing so.

Single Thread Scanning on Host

Sandfly is written in Go. Go is a high performance memory safe multithreaded programming language designed by Google. While the backend systems are multithreaded, our on-host investigation scanning engine deliberately does not multithread. There are several reasons we do not multithread this component:

1) Investigation scans are fast enough for a single core.
2) Eliminates potential bugs coordinating threads.
3) A runaway scan can only use 100% of a single core on a multicore system.

The third point is the most salient. Even if all our failsafe mechanisms fail to shut down a scan, the worst that happens is a single core is consumed until the scan completes. On a modern processor this leaves remaining cores to service other tasks without consequence.

Low Process Priority (Nice Level)

Sandfly uses a medium-low priority (nice level) by default. Manual scans may run at a higher priority if selected by the user. The API allows full control of what priority level to run any scan. Low priority is more than sufficient for most scans without causing noticeable system load.

Data Caching

Sandfly makes intelligent use of caching. For instance, instead of scanning the process table many times, we build a process table cache for each session. All Sandfly modules run against this cache. We do the same for file, user, and other forensic attributes.

Memory

Sandfly can run on systems ranging from over decade-old, obsolescent hardware to modern cloud and embedded devices. Sandfly also runs well on memory constrained environments.

Memory Caching Limits

We use caching inside the Sandfly binary for performance, but we limit the maximum size of a cache so that memory usage is controlled.

Memory Safety and Efficiency

The Go language is well-known for memory safety and efficiency. We have tested our binary on the following CPU and memory constrained systems with no issues:

- AMD64 Low End Shared CPU VM w/1GB RAM
- Arm7 Embedded System w/256MB RAM
- MIPS LE Power over Ethernet (PoE) Camera w/256MB RAM

Test results		
System Type	**Avail Memory**	**Used Memory**
AMD64 Shared CPU VM	*1GB*	*130MB**
Arm7 Embedded	*256MB*	*55MB**
MIPS LE PoE Embedded	*256MB*	*52MB**

*** Note that free memory is used efficiently by Linux and these values will shrink considerably if the remote system has less free RAM available.**

Disk

Disk I/O is often the most crucial bottleneck for performance. For instance, it is possible to have a 64 core CPU at low utilization. But, if disk I/O is at 100% capacity, the entire system will perform poorly no matter how much CPU headroom is present. Sandfly takes special care to ensure we do not cause large disk impacts.

Minimal Disk Write

Sandfly does not write to the disk except to upload the Sandfly binary. We can also optionally write to ramdisk to preserve embedded and write-limited storage (such as SD cards). The binary is deleted after the scan is run to not consume drive space.

Disk Performance Impacts

The most resource intensive Sandfly scans are file and directory checks. Internally, Sandfly does the following:

- Limits recursive checks to not sweep the entire, or even large parts, of a disk.
- Does not perform legacy signature checks of files (e.g. massive anti-virus/malware scans), which is not only data intensive, but works poorly on evolving Linux malware.
- Limits search depth on files so we do not waste time crawling large blocks of data.
- Caching to not repeat expensive operations such as hash or entropy calculations.

The most expensive file checks are those calculating cryptographic hashes and file entropy. Both of these operations require reading in entire files to calculate these values. Our default checks limit the number of directories we will scan for this data and we use caching as appropriate to ensure we don't calculate these values unless needed.

Other checks involving incident response or "reconnaissance" checks that pull file attributes for drift detection can have varied impact. If a customer scans the top level root file system, for instance, they can easily profile millions of files on a Linux server. This will cause both high disk I/O and high CPU loads on a single core. For this reason we recommend customers to avoid these operations unless they are actively investigating an incident.

Network

Sandfly passes network traffic to and from remote endpoints in two ways:

1) Sending our investigation binary to the system to execute security sweeps.
2) Retrieving results from an endpoint for analysis and presentation.

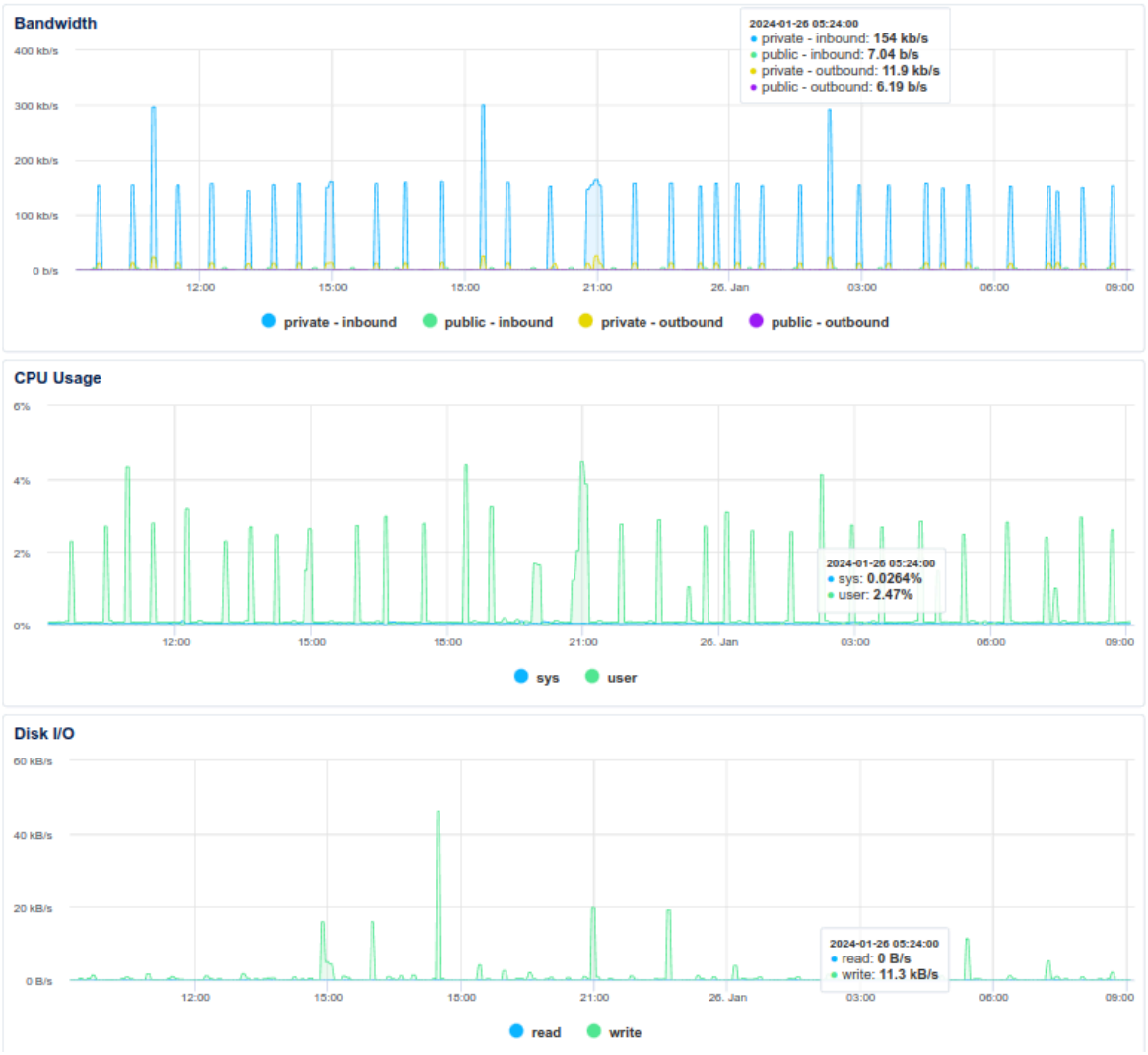
Inbound, the Sandfly binary is about 6MB in size (smaller than many JPEGs). It is pushed over during the initial connection and removes itself when done.

Outbound, alert data is compressed and is reduced in size by about 90% when sent. This results in significant network data savings and speed. Network I/O can be further limited by reducing the frequency and number of checks Sandfly does each time we connect to a host. Sandfly runs perfectly fine on mobile data connections with limited bandwidth.

Performance Graphs

Below is a 24 hour graph from a test host with 1GB of RAM on a low-end shared CPU VM. We deliberately use the cheapest commercially available shared CPU virtual host with low RAM for tests. It's the marginal systems where we want to make sure we don't cause impacts.

The irregular spikes are the random schedules of Sandfly scans. Average CPU usage when Sandfly runs is in the 2-4% range with brief spikes to 100%. For this test we are running a heavier than default Sandfly scan each time (65-100% Sandfly module selection) to show a worst case. CPU, network, and disk I/O are well contained and not a bottleneck.



Fail-safes

Sandfly has multiple fail-safes built-in to ensure scans do not take too long or malfunction.

Individual Sandfly Timeout

Each module has an individual value on how long it may run before being stopped by the Sandfly engine. The default value is typically 360 seconds (6 minutes) with a maximum of 1800 seconds (30 minutes). Sandflies exceeding the individual timeout value are stopped and reported as a timeout error. The next Sandfly module is then run normally.

Group Sandfly Timeout

If two individual Sandfly modules timeout in a session, we will halt the entire scan. This avoids overwhelming a remote system that is likely overloaded with tasks unrelated to Sandfly. Two modules timing out will generate a full scan stop error in the Sandfly error log.

Global Timeout at Node

As a final check, the scanning nodes implement a global timeout, which defaults to one hour if the scan hasn't completed for that system. This global timeout ensures that a system with unknown problems does not continue to run scans even if very slow.

Recommendations

Overall, Sandfly takes great care to not cause stability risks nor to not overwhelm systems. Customers looking to lower system impacts further may consider the following:

- Do not run intensive incident response sandflies except as needed.
- Limit file and directory checks to a lower frequency schedule.
- Lower Sandfly random percentage selection values on scheduled scans.

Customers running Sandfly have always noted our significantly lower system performance impacts vs. conventional agent-based products. We take pride in knowing we can protect virtually any Linux system with little risk to the host. If you have any further questions about our safety, performance, or compatibility, please reach out to our team.